Assignment 3_4 Report

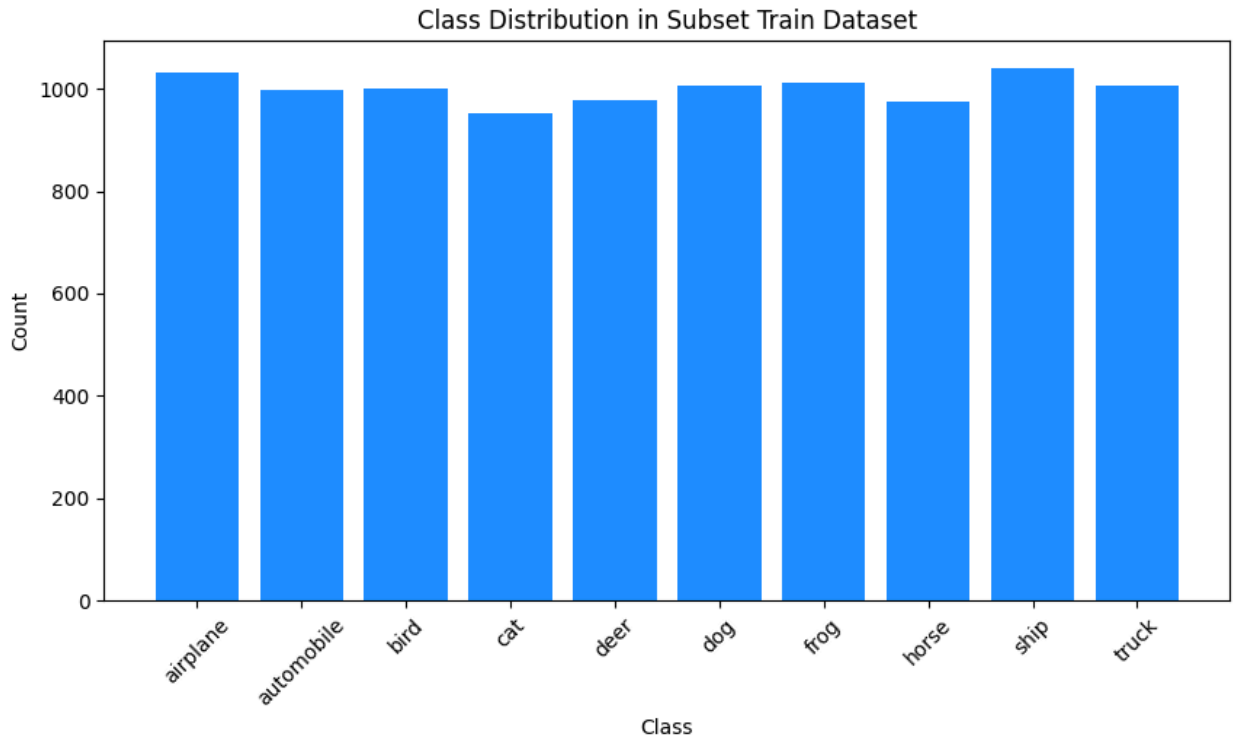
AI at Scale Lab - ID5003W

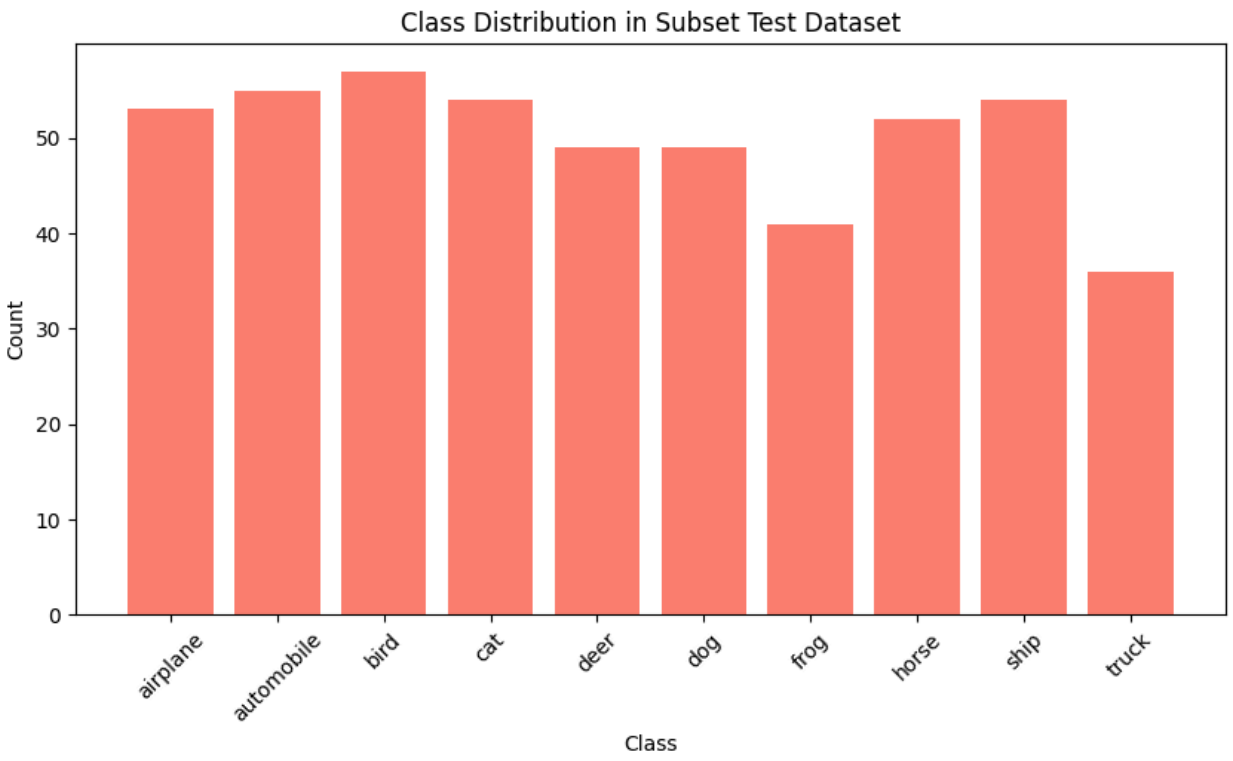
SOUMYA MUKHERJEE | CH24M571

Dataset in question : CIFAR10

Part - 1 :

Class distribution in the complete train and test datasets





Part 2 Question 2 :
The classifiers present are 'Classifier', 'DecisionTreeClassifier', 'FMClassifier', 'GBTClassifier', 'MultilayerPerceptronClassifier', 'ProbabilisticClassifier', 'RandomForestClassifier'

I took Logistic Regression , Random Forest and Naive Bayes . The 3 models are , in my opinion, 3 good baselines with 3 different paradigms - linear , non-linear and probabilistic approaches .

Logistic regression was used to see if the data was linearly separable. It is efficient on high dimensional but not too large datasets

Random forest captures non-linear relationships and feature interactions automatically . It is robust to noise to certain degree

NaiveBayes is a probabilistic model and it assumes feature independence . Since it is much simpler and less resource intensive , it gives a good speed - accuracy tradeoff

Comparison of running time and metrics

Configurations Tested:

Config 1 → executor_cores=1, max_cores=2, executor_memory=4g

Config 2 → executor_cores=2, max_cores=4, executor_memory=6g

Config 3 → executor_cores=4, max_cores=6, executor_memory=8g

—----------

Config	Model	Accuracy	F1-score	Time (s)
1 (1c-2m-4g)	Logistic Regression	**0.8720**	**0.8726**	17.14
	Random Forest	0.7080	0.7034	13.58
	Naive Bayes	0.7960	0.7961	**4.01**
2 (2c-4m-6g)	Logistic Regression	**0.8720**	**0.8726**	9.24
	Random Forest	0.7260	0.7206	8.12
	Naive Bayes	0.7960	0.7961	**2.31**
3 (4c-6m-8g)	Logistic Regression	**0.8720**	**0.8726**	9.22
	Random Forest	0.6880	0.6822	8.33
	Naive Bayes	0.7960	0.7961	**2.15**

Parallelism Effect

Config 1 [executor_cores=1, max_cores=2, executor_memory=4g] had the longest training times due to limited cores, though all models completed successfully.

Config 2 [executor_cores=2, max_cores=4, executor_memory=6g] nearly halved Logistic Regression's training time compared to Config 1, and Random Forest also benefited significantly.

Config 3 [executor_cores=4, max_cores=6, executor_memory=8g] provided additional cores but did not improve performance further; training times for Logistic Regression and Random Forest plateaued around ~9s and ~8s, respectively. This indicates that beyond a certain point, more

cores do not yield significant speedups, likely due to overhead and the relatively small dataset size.

Model Performance

Logistic Regression consistently achieved the highest accuracy and F1-score (0.872) across all configs, making it the most reliable model.

Random Forest performed worse in terms of accuracy (0.68–0.72) but had moderate training time. Interestingly, accuracy decreased with more parallelism (Config 3).

Naive Bayes consistently trained the fastest (2–4 seconds) but produced lower accuracy (0.796) than Logistic Regression.

Resource Utilization

Config 1 underutilized available resources, leading to unnecessarily long runtimes.

Config 2 struck a good balance between cores and memory, showing the best efficiency.

Config 3 did not improve results further and even caused Random Forest accuracy to drop, possibly due to Spark's execution overhead.

Config 2 (`executor_cores=2, max_cores=4, executor_memory=6g`) was the most efficient. It achieved near-minimal training times without over-allocating resources. Config 3 showed diminishing returns.

MobileNetV2 feature embeddings are already high-level, compressed, and fairly linearly separable which is likely the reason why Logistic Regression worked so well.

Question 3 : BigDL Orca

Details on my model choice

1. *Convolutional Blocks*

Conv2d (3→32, then 32→64):
Two convolutional layers progressively expand feature channels (32, 64), enabling the model to learn low- and mid-level visual features (edges, textures, shapes).

BatchNorm2d: Applied after each convolution to stabilize training, control internal covariate shift, and allow higher learning rates.

LeakyReLU: Introduced instead of plain ReLU to avoid the “dying ReLU” problem and improve gradient flow for negative activations.

MaxPool2d (stride 2): Reduces spatial resolution (32×32 → 16×16 → 8×8), focusing on essential patterns while lowering computational cost.

Dropout (p=0.25): Applied after pooling in both blocks to regularize the model, mitigate overfitting, and improve generalization.

2. *Global Feature Aggregation*

AdaptiveAvgPool2d(1): Replaces large fully connected layers by computing global average pooling (8×8 feature maps → 1×1 per channel).

Significantly reduces parameters (64 → 10 linear layers only).

Encourages class activation maps to focus on discriminative features.

Improves robustness compared to dense layers.

3. *Classification Head*

Linear(64→10): Final projection to 10 CIFAR-10 classes. CrossEntropyLoss directly consumes logits, hence no explicit softmax is applied in the model.

2 important points I have kept in mind

-> Regularization – Dropout + BatchNorm provide resilience against overfitting.

-> Stability – LeakyReLU and BatchNorm support faster, more stable convergence.

—----------

Discussion on the output result

Q2 Results (Parallelism Comparison, 10k Training Samples)

Cores	Train Loss (final epoch)	Validation Accuracy	Validation Loss
2	1.60	**43.55%**	1.58
4	1.60	**44.50% (best)**	1.59
8	1.61	42.50%	1.59

—----------

Results and Discussion (Q2: Parallelism Comparison)

The effect of parallelism was evaluated by training the model on a 10,000-image subset of CIFAR-10 with different core allocations (2, 4, and 8). Across all settings, the training loss decreased consistently from ~2.0 at epoch 1 to ~1.6 at epoch 5, confirming stable convergence under distributed execution.

Validation results showed the highest accuracy at 44.50% with 4 cores, compared to 43.55% with 2 cores and 42.50% with 8 cores. Validation loss remained nearly identical across all configurations (~1.58–1.59), suggesting that generalization capability was unaffected by parallelism and that observed differences arose mainly from training efficiency.

The decline in accuracy at 8 cores likely reflects parallelization overheads outweighing benefits at this relatively modest dataset size. Thus, 4 cores represented the optimal trade-off, achieving the best validation accuracy while maintaining efficient utilization of computational resources. This configuration was subsequently adopted for larger-scale training in Q3.

Observations :

- The loss goes down smoothly and accuracy improves .
- The model is underfitting — both train and validation losses are still relatively high, and train accuracy is not near 100%.
- Since we have only 2 convolutional blocks it is too shallow to extract complex hierarchies of features.
- Since we have trained for only 5 epochs , it's not enough time to converge properly

Random guessing would have given 10% accuracy , whereas our simple CNN has given 43-46% accuracy, which is a good starting point .